Science Search and Retrieval using XML

Daniel Crichton, Steven Hughes, Jason Hyon, Sean Kelly
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109

Abstract - Science missions and instruments continue to produce volumes of useful data and scientists depend on the data systems and tools that archive this data as a means to access and analyze it. These existing legacy systems do not interoperate well, and scientists must access each data system and its corresponding science data independently through tools that have been custom-built for the particular science data system or mission. The Object Oriented Data Technology task is working on the distributed resource location service, which will allow location and exchange of geographically distributed data. Advances in Internet and distributed object technologies provide an excellent framework for sharing data across multiple data systems. The Extensible Markup Language (XML) and the Common Object Request Broker Architecture (CORBA) provide support for electronic data interchange (EDI) between heterogeneous data sources. CORBA provides the over-the-wire exchange of XML-based profiles that contain descriptive information of science products archived at remote data systems. This paper discusses a framework for data system interoperability that will not only benefit space science, but provide a cross-disciplinary solution for a next generation data system architecture.

I. Introduction

Science data has continued to devolve into a large set of highly fragmented distributed data systems. These systems are heterogeneous and geographically distributed making interoperability and integration difficult. Furthermore, correlating science data across a multi-disciplinary environment is even more challenging. The Object Oriented Data Technology task at the Jet Propulsion Laboratory is currently researching a distributed framework that will allow for dataset resources and products to be exchanged based on a set of distributed systems called the distributed resource location service.

The distributed resource location service enables applications to locate geographically distributed science data in heterogeneous data systems without knowing which data systems and catalogs to search, or what the interfaces are to each catalog. The resource location service manages a hierarchical conglomerate of dataset resource definitions that allow for data products residing in distributed data systems to be located. The intent of the hierarchical view is that clusters of data systems may be organized as sub-components of larger communities. For example, NASA's Office of Space Science has

hundreds of data systems each containing numerous datasets and catalogs that make up the multi-disciplinary communities of planetary, astrophysics and space physics science data. Within each community, there can exist smaller communities that can point to data systems, products, or even other communities. Once data is located, the service converts it to a neutral format to enable correlation across science data systems.

II. Architecture

The resource location service has several architecture objectives. These include (1) requiring that individual data systems be encapsulated to hide uniqueness; (2) requiring that communication between distributed services use metadata for data interchange; (3) defining a standard data dictionary based on a metadata for describing data resources; and (4) providing a solution that is both scalable and extensible.

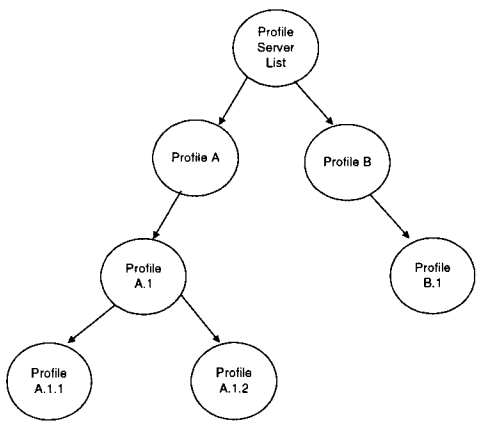

Figure 1: Distributed profile architecture.

The resource location service centers around an architecture based on a directed graph[1] (or digraph) of resources that are traversed in order to satisfy a query. Profiles—sets of resource definitions—describe nodes of the digraph. Profiles may point to other profiles thus representing arcs of the digraph. A profile is essentially a metadata description of the resources known at a node in the distributed framework. These resources are either data products archived by an integrated data system, or definitions of other profile nodes that manage metadata about other data systems that can further satisfy the query. Figure 1 depicts an example of a digraph of profiles that represents a set of distributed data systems.

The resource location framework consists of three components that include a query service component, a profile service component, and a product service component. The components provide the functionality necessary to traverse the profile digraph and return products for located resources. The query component executes concurrent queries to the profile and product components in order to satisfy a query. Profile components manage a set of profiles (or resource definitions) for a particular node in the digraph. The product component then provides the translation necessary to map a product retrieved from a data-system–dependent environment into a neutral format suitable for exchange between systems.

Each node of the architecture exchanges data based on metadata definitions. These definitions define how data is queried and returned, as well as how the profiles are encoded at each node. Product components are similar to profile components in that they also represent a set of distributed nodes. A product node wraps the interface to one or more data systems so as to sanitize the data and map it into a standard format that can be exchanged across the architecture. This allows heterogeneous data systems to be easily added without changing the way their data is stored.

The component architecture described lends itself naturally to a distributed object implementation. We used the Common Object Request Broker Architecture (CORBA) to provide the distributed object framework and to communicate and exchange data in heterogeneous environments using the Internet Inter-ORB Protocol (IIOP). This activity is currently using an implementation of the CORBA 2.0 standard from Object Oriented Concepts known as Orbacus. Each profile and product server node is defined by a separate object name (or node name). CORBA allows for nodes to be located based on the CORBA naming service that is included in the Orbacus implementation. The naming service allows objects that can satisfy the query to be specified by name so that profiles can identify other

[1] A directed graph consists of a set of vertices V and a set of arcs E. Vertices are also called nodes or points; Arcs are also known as directed edges or directed lines.

profiles or products in their metadata definitions. This enables integration of the described nodes.

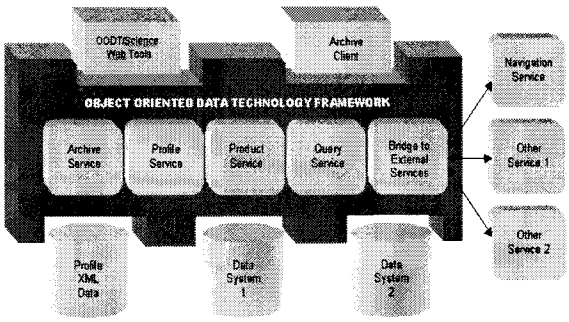

Figure 2: Component framework

Each component of the architecture communicates with other components using the Extensible Markup Language (XML) for the data content running on top of the CORBA implementation. One of the critical requirements of this architecture is to provide interoperability solutions without having to change the implementation of each data system. Our architecture accomplishes this goal by encapsulating each of the individual data systems and then using standard metadata definitions based on XML for interoperability. This allows various implementations ranging from the use of relational and object database management systems to implementations that use flat file and home-grown databases for cataloging and storing data products to exchange information using XML metadata definitions.

We deployed the resource location service entirely in the Java programming language along with CORBA and XML. Java allows for the implementation of the object architecture and allows the framework to be easily extended to integrate new data systems. Java is particularly useful in the design of the product service component which allows new servers to be quickly instantiated by loading additional product translation objects at run time. This will be explained in further detail below.

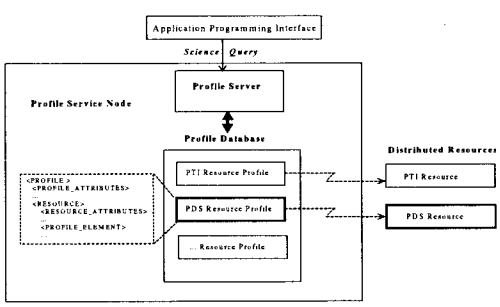

Figure 3: Profile Service Node

One of the goals of this architecture is to provide a standard application programmers' interface (API) that will allow for generic science analysis tools to be written that can plug into the architecture to retrieve and correlate data from multiple data sources. This is accomplished using an *n*-tier architecture. Such architectures split the traditional client-server model into three layers: a user interface layer, a domain logic layer, and a storage layer. Abstracting the implementation away from the client allows for the infrastructure to evolve without breaking the tool interfaces. It also moves the domain intelligence to the middleware components which removes the constraint that the tools need to have the knowledge of the protocol and location of data systems in order to query and retrieve data from it. Finally, the *n*-tier architecture also allows the framework to plug in additional services.

In Figure 2 the framework shows general objects that fit into the framework along with bridges to other services that could be potentially added. In this case, a navigation service could be added to allow for images of Jupiter, a constantly moving target, to be found based on metadata that describes the right ascension and declination (RA/DEC) of an image. Also since the navigation service performs coordinate system transformation, this addition illustrates how a software component can use metadata to further increase interoperability between domains.

Figure 3 illustrates the functioning of one profile service node within the resource location framework. The profile server node is managing resource profiles from multiple disciplines, namely the Palomar Testbed Interferometer (PTI), an astrophysics system, and the Planetary Data System (PDS). A candidate query for an image from the Mars Global Surveyor mission arrives at the node from the API. A candidate PDS resource is then identified by searching the resource profile database. The query system will subsequently use a PDS product delivery service to obtain product information from the resource. Product information may include images, time series data, or simply metadata information. The product delivery service will be described in a subsequent section of the paper.

The component architecture as described focuses on providing a framework for solving complex

integration problems across heterogeneous data systems. It addresses the issues of data location, data transformation, and data exchange. The framework provides a scalable architecture that centers around the use of metadata. It also allows for data systems to continue to retain their unique attributes, yet plug into an enterprise architecture that allows for the successful exchange of data content through the use of XML. By using XML this framework is able to impose an inter-disciplinary communication mechanism that allows for data to be shared and exchanged.

III. Query Service

The query component of the framework serves as the starting point for users to retrieve information stored across distributed data nodes. The query component's CORBA interface enables analysis tools to have a programmatic entry point for entering queries and retrieving results. In addition, we have implemented a Java API that wraps the CORBA interface (a C++ API is forthcoming). This enables scientists and engineers to develop their own data analysis tools to access disparate data systems from a single API. As more data systems are added to the framework, existing tools can access the new systems with no changes. Furthermore, multiple user interfaces that access the query component are possible. One such interface that we have developed is a web interface. The web interface uses the Java API to give scientists and engineers immediate access to data systems from any common web browser without any programming or knowledge of what data systems to search.

The query service uses the CORBA naming service to connect to a profile node. In general, searches will enter the directed graph at the root or parent node; however, the query service can enter and search at any point in the graph. Profiles must be registered in order to be searched.

To execute a search, the query component assembles an XML document describing the characteristics of the query. The document includes a header section that describes metadata about the query, such as its title, description, data dictionary, security type, and revision code. These elements indicate to the query service any characteristics, versioning, or special handling required by the query. Also included in the document are preferences on the result, such as how the query should propagate through the digraph, the maximum number of results, the query itself, and a space for results. For example, a query for "TARGET_ID=MARS" results in the

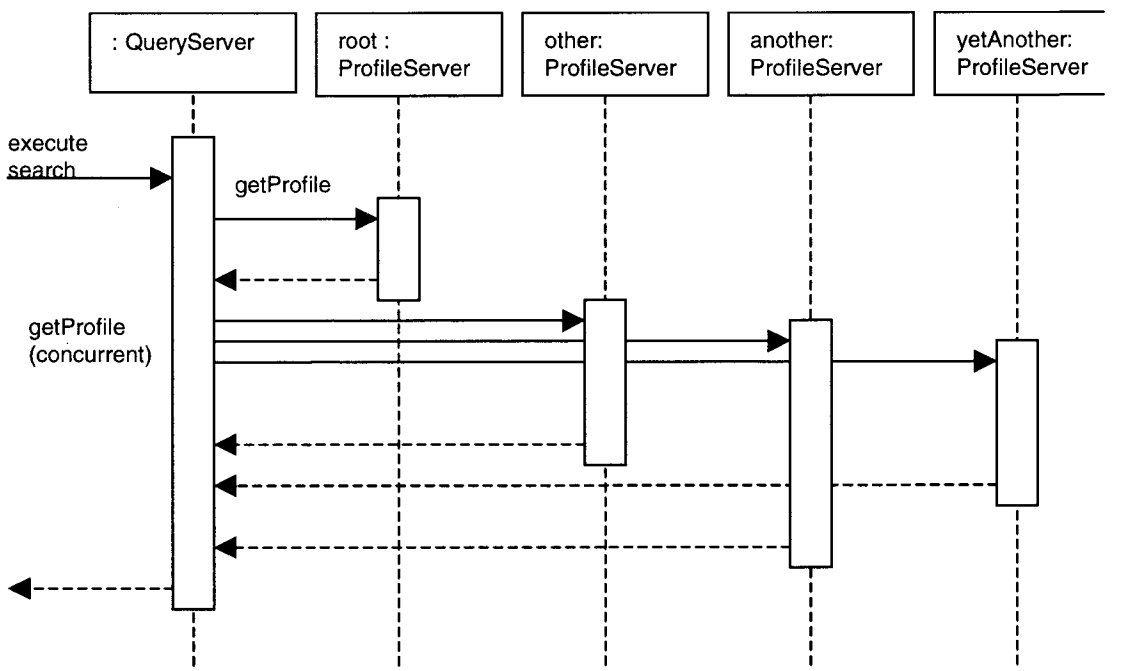

Figure 4: UML Diagram for Profile Search

following XML document:

```
<?xml version="1.0"?>
<QUERY>
  <QUERY_ATTRIBUTES>
    <ID>OODT_XML_QUERY_V0.1</ID>
    <TITLE>OODT_XML_QUERY Example</TITLE>
    <DESC>This query can be handled by PDSDIS</DESC>
    <TYPE>QUERY</TYPE>
    <STATUS_ID>ACTIVE</STATUS_ID>
    <SECURITY_TYPE>UNKNOWN</SECURITY_TYPE>
    <REVISION_NOTE>1999-12-12 JSH V1.0</REVISION_NOTE>
    <DATA_DICTIONARY_ID>OODT_DATA_SET_V1.0
    </DATA_DICTIONARY_ID>
  </QUERY_ATTRIBUTES>
  <RESULT_MODE_ID>ATTRIBUTE</RESULT_MODE_ID>
  <PROPOGATION_TYPE>BROADCAST</PROPOGATION_TYPE>
  <PROPOGATION_LEVELS>N/A</PROPOGATION_LEVELS>
  <MAXIMUM_RESULTS>100</MAXIMUM_RESULTS>
  <RESULTS>0</RESULTS>
  <KWQ_STRING>DATA_SET_NAME=MARS</KWQ_STRING>
  <QUERY_SELECT_SET/>
  <QUERY_FROM_SET/>
  <QUERY_WHERE_SET/>
  <QUERY_RESULT_SET/>
</QUERY>
```

The query service "crawls" through multiple nodes in the directed graph of resource systems automatically, locating additional servers that can fulfill a request for a particular item in any number of datasets. The query service uses "spider" objects to execute queries on each profile's XML description. The spider objects are part of the scatter-gather approach: each object can run in its own thread of execution, maximizing the concurrency of multiple nodes in the system. The system scatters the spiders across nodes and gathers their results as they become available.

Figure 4 shows a Unified Modeling Language (UML) sequence diagram for a typical search. In the diagram, objects are shown across the top with their lifelines dropping down as time increases. Rectangles over the lifelines depict when an object is active. Solid arrows show method calls on an object, while dashed arrows show returns from those calls. A user's query triggers the action at the Query Server object through its "execute search" method. The Query Server asks its root Profile Server object for any matches to the query. In response, the Profile Server returns three possible other Profile Servers that could contain matches (in addition to any dataset matches it itself has). Concurrently, the Query Server executes the same query on the other Profile Servers. As each server returns more information, the Query Server may query yet more and more servers. Finally, after traversing the digraph in this way, the Query Server returns the search results in an XML document.

The Java programming language simplifies development of concurrent programming such as that used in the query component. Java includes built-in keywords and library classes for threading. However, Sun's marketing phrase for Java, "Write Once, Run Anywhere," is more hype than reality. In order to encourage implementations of Java on a wide variety of computer hardware and operating systems, Sun underspecified details of Java's threading behavior. As a result, behavior of threaded programs vary from implementation to implementation. Sometimes, programs hang (deadlock) even though there is no obvious deadlock in the code as implemented.

The query service experiences such hanging behavior. When running in a Windows-based Java environment, multiple concurrent queries work correctly and quickly. But on a Linux-based environment, multiple threads performing queries hang the query component. Because the scatter-gather approach to making multiple concurrent queries is far more efficient than serially querying remote nodes, we plan on incorporating a deeper investigation of the code and of various Java virtual machines for the Linux platform.

Since the directed graph of resource systems is not necessarily acyclic, the query component must take care not to re-query profile nodes it has already visited, or else it could get caught in an infinite loop. The query component trivially prevents this by tracking a set of profiles it has queried so far.

Once the query component's spiders have completed their tasks, the query service assembles the results into an XML document. The user can access the results document directly or it can be translated into HTML for presentation within a web browser. In the web browser, hyperlinks and additional searches are set up automatically by the translation process that enables the end user to immediately fetch products or visit sites that contain the sought datasets.

One possible extension that we are considering for the query service is to make it available via the HTTP standard. This would allow HTML pages to send XML queries through the resource location service and render results directly into the HTML document as previously mentioned.

IV. Profile Service

Science data systems contain product results. Instruments and experiments generate results that are archived into heterogeneous data systems. Unfortunately, there is no standard for querying these data systems for their content and it makes locating data nearly impossible. Scientists and researchers are currently required to visit each data system independently and use tools that are unique to the data system in order to locate information. The profile service that is part of the framework uses metadata[2] to describe a variety of information about data resources that can exist within a distributed data environment. It refocuses the problem of interoperability on metadata development and enables interoperability by using a common metadata interchange language.

The purpose of an OODT profile is to provide a resource description, or metadata, that is sufficient to determine if the resource can resolve a query. It is used by the OODT resource location service to identify and locate resources within the digraph and subsequently limit the number of resources that will have to consider the query. For example, within a space sciences implementation of this concept, a query for images of Jupiter taken by the Hubble Space Telescope should not have to be handled by the resource maintaining the Mars Global Surveyor spacecraft images of Mars. A profile can be defined a proper subset of the metadata that describes a resource and as stated about, that is sufficient to determine whether the resource could resolve a query.

The OODT profile development effort had several phases. The first phase was to decide what language should be used to manage the metadata. Since the underlying requirement was the need for a common interchange language, we identified the Extensible Markup Language (XML) as being ideally suited to the problem. The advantages of XML include (1) superior expressiveness to HTML by allowing information-structure specifications, (2) simplicity compared to SGML in use and syntax, and (3) wide acceptance as an Electronic Data Interchange (EDI) standard. However the most compelling aspect of XML is that even though it can be used to capture metadata by directly mapping data elements to tags, XML can also be used as a meta-language to define a language. This allowed us to develop a generic language for managing metadata from any domain.

The second phase of the profile development effort was to develop a generic structure for capturing metadata for resources from disparate domains. We used XML to define the XML Extensible Profile Language (X2PL). X2PL provides both a means for capturing resource attributes as well as a language for capturing the attributes of the information content that the resource manages. The Document Type Definition (DTD) specification in Figure 5 illustrates the basic components of the resource profile. The DTD specification consists of three sections: the profile attributes, resource attributes, and the profile elements. We're using a DTD specification since the technology is readily understood and widely supported. We will consider more powerful specification mechanisms such as XML-SCHEMA once they have been approved as a standard by W3C.

The profile, as an object itself, is described in the profile attributes section using the attributes shown. The ID attribute provides a system-wide unique identifier for the profile instance. The TITLE and DESC attributes provide descriptions of the profile, with the TITLE being more terse and appropriate for frequent display. The TYPE attribute, defined below, identifies a subtype. The DATA_DICTIONARY_ID attribute provides the identifier of a controlling domain data dictionary and contains addition information that might not be appropriate at the profile level. The CHILD_ID and PARENT_ID attributes provide the identifiers of related profiles and allow for the creation of a hierarchy of profiles.

The resource description starts in the second section of the profile, using the attributes shown in figure 5. The RESOURCE_ID and RESOURCE_TITLE attributes are analogous to the profile ID and TITLE, providing identification and descriptive information. The RESOURCE_DISCIPLINE attribute identifies the discipline within which the resource exists and is taken from a discipline taxonomy. For example, planetary science is a discipline within space sciences and itself consists of several

[2] Metadata is, literally, data about data, or information that describes the characteristics of data. For example, 37.6 is data. The fact that it's a measurement of a body's temperature in Kelvins is metadata.

disciplines such as Geosciences, Planetary Plasma Interactions, and Atmospheres. The RESOURCE_LOCATION_ID and RESOURCE_MIME_TYPE attributes provide the location of the resource and the MIME type of the resource's response, respectively. The RESOURCE_AGGREGATION attribute indicates the data aggregation level managed by the resource. These levels are GRANULE or individual data products, GRANULE+ or data production collections (data sets), and GRANULE++ or data set collections. The RESOURCE_CLASS attribute is used to locate the resource within a resource taxonomy. Examples are PRODUCT_SERVER, PROFILE_SERVER, CATALOG, INVENTORY, and INTERFACE.

OODT Profile Document Type Definition (DTD)

```
<!ELEMENT PROFILES
 (PROFILE+)>

<!ELEMENT PROFILE
 (PROFILE_ATTRIBUTES,
  RESOURCE)>

 <!ELEMENT PROFILE_ATTRIBUTES
  (ID, TITLE*, DESC*, TYPE*,
  STATUS_ID*, SECURITY_TYPE*,
  PARENT_ID*, CHILD_ID*,
  REVISION_NOTE*,
  DATA_DICTIONARY_ID*)>

<!ELEMENT RESOURCE
  (RESOURCE_ATTRIBUTES,
  PROFILE_ELEMENT*)>

<!ELEMENT RESOURCE_ATTRIBUTES
   (RESOURCE_ID,
   RESOURCE_TITLE,
   RESOURCE_DISCIPLINE,
   RESOURCE_AGGREGATION,
   RESOURCE_CLASS,
   RESOURCE_LOCATION_ID,
   RESULT_MIME_TYPE)>

<!ELEMENT PROFILE_ELEMENT
(ELEMENT_NAME,
ELEMENT_MEANING*,
ELEMENT_ALIAS*,
VALUE_SYNTAX*,
VALUE_UNIT*,
(VALUE_INSTANCE |
(MINIMUM_VALUE,
MAXIMUM_VALUE))*)>
```

Figure 5: OODT DTD

The profile element section is the third part of the profile and provides the second part of the resource description by describing the information content that the resource manages. For example, within the planetary science community, the Planetary Data System (PDS), maintains the Distributed Inventory System (DIS), an inventory of all science data sets that have been archived in the system. Within the inventory, the data sets are indexed on the associated spacecraft instrument and target body. The profile element section of a resource profile for the DIS would include these indexed attributes as data elements. Figure 6 shows a portion of the DIS resource profile.

As can be seen from Figure 5, each data element is defined in the profile structure using meta-attributes such as ELEMENT_NAME and VALUE_INSTANCE. The data elements defined in the profile element section are specific to the discipline identified in the RESOURCE_DISCIPLINE and are also defined in the controlling data dictionary referenced by DATA_DICTIONARY_ID. To maintain compliance with international standards, these meta-attributes are consistent with those defined in the Data Entity Dictionary Specification Language (DEDSL) (2), and are briefly described here. The ELEMENT_NAME attribute provides a unique identifier for the data element definition in the data dictionary. The VALUE_SYNTAX attribute provides the encoding type for the data element. The VALUE_INSTANCE, MINIMUM_VALUE, and MAXIMUM_VALUE attributes provide either preferred values for enumerated data types or the upper and lower bounds for numeric data types. The ELEMENT_MEANING attribute provides a textual description of the data element and the ELEMENT_ALIAS attribute identifies synonyms.

The third phase of the profile development effort focused on specializing the profile to meet three slightly different sets of requirements. As is apparent, the profile element section is essentially a data dictionary in that it provides data element definitions. Because of this fact and the need to reduce complexity, the profile structure was specialized to into three subtypes: PROFILE, DATA_DICTIONARY, and INVENTORY.

The PROFILE specialization is consistent with what has been described above and is used primarily when one profile is required for one resource. The specialization specifically makes the use of the meta-attributes ELEMENT_MEANING and ELEMENT_ALIAS optional. For example, in Figure 6 PDS DIS attributes from the planetary science domain are defined as data elements in the profile element section. Using this information, the query service is able to determine that a query with a constraint of "TARGET_NAME = IDA" could be sent to this resource. Since a common vocabulary is being used, the assumption is made that the resource will be able to resolve the query.

The next specialization of the profile structure is DATA_DICTIONARY. For a DATA_DICTIONARY, the profile element section defines the attributes of any object managed by any resource in the domain, focusing primarily on those attributes used for indexing. In addition, the meta-attributes ELEMENT_ALIAS and ELEMENT_MEANING are considered more

```
<PROFILE ...
...
<RESOURCE>
  <RESOURCE_ATTRIBUTES>
    <RESOURCE_ID> PDS_MGS_MOC_IMAGE_nnn </RESOURCE_ID>
    ...
  </RESOURCE_ATTRIBUTES>
  ...
  <PROFILE_ELEMENT>
    <ELEMENT_NAME> TARGET_NAME </ELEMENT_NAME>
    <VALUE_INSTANCE> MARS </VALUE_INSTANCE>
  </PROFILE_ELEMENT>

  <PROFILE_ELEMENT>
     <ELEMENT_NAME> INSTRUMENT_HOST_ID </ELEMENT_NAME>
     <VALUE_INSTANCE> MGS </VALUE_INSTANCE>
  </PROFILE_ELEMENT>

  <PROFILE_ELEMENT>
     <ELEMENT_NAME> INSTRUMENT_ID </ELEMENT_NAME>
     <VALUE_INSTANCE> MOC </VALUE_INSTANCE>
  </PROFILE_ELEMENT>
  ...
```

Figure 6: Example Profile

significant and the preferred values, for any enumerated types are the union of all preferred values over the domain. For example, in the data dictionary for the planetary science community, the TARGET_NAME data element would have names of all planets, satellites, comets, and asteroids for values of VALUE_INSTANCE.

The final specialization of the profile structure is the INVENTORY. This profile subtype is a slight variation on PROFILE and is used when one profile structure is used to describe many resources. For example, one INVENTORY could be used to profile all the Mars Global Surveyor images of Mars. This specialization minimizes space by requiring only one set of profile attributes and by reducing the number of required meta-attributes. For example, there is no need to provide the ELEMENT_MEANING for TARGET_NAME for each image.

Additionally, the preferred values of data elements are also constrained. For example, the values of TARGET_NAME for any image in this data set would have a single valid value of MARS. Figure 7 illustrates a portion of an INVENTORY profile for one such image.

The final phase of the profile development effort involves implementing instances of profiles for specific domains. As is evident, the success of the distributed resource location concept is dependent on the existence of domain metadata captured in repositories such as data (element) dictionaries. Within such privileged domains, the registration of resources with the service is readily accomplished by extracting the necessary metadata from the domain's metadata repository, creating the resource profile, and then registering the profile with the service. This enables the successful location of resources within a domain, possible location of resources across domains, and even raises the possibility of resource interoperability.

The PDS resource profiles shown in Figures 6 and 7 were generated by extracting metadata from the PDS DIS. Metadata from other disciplines have been selected from relational

Profile Example - PDS Distributed Inventory System

```
<PROFILE PROFILE_ID = "PROFILE_PDS_DIS_V1.3.n" >
  <PROFILE_ATTRIBUTES>
    <ID> PROFILE_PDS_DIS_V1.3.n </ID>
    <TITLE> Planetary Data System - Distributed Inventory System - Profile V1.0 ...
    <DESC> This profile describes the Planetary Data System (PDS) Distributed ...
    <TYPE> PROFILE </TYPE>
    <DATA_DICTIONARY_ID> OODT_PDS_DATA_SET_DD_V1.0 ...
  </PROFILE_ATTRIBUTES>

  <RESOURCE>
    <RESOURCE_ATTRIBUTES>
      <RESOURCE_ID> PDS_DIS_V1.3.n </RESOURCE_ID>
      <RESOURCE_TITLE> Planetary Data System - Distributed Inventory System ...
      <RESOURCE_DISCIPLINE> PDS </RESOURCE_DISCIPLINE>
      <RESOURCE_AGGREGATION> GRANULE+ </RESOURCE_AGGREGATION>
      <RESOURCE_CLASS> INVENTORY </RESOURCE_CLASS>
      <RESOURCE_LOCATION_ID> http://pds.jpl.nasa.gov/pdsbrows.htm ...
      <RESULT_MIME_TYPE> text/html </RESULT_MIME_TYPE>
    </RESOURCE_ATTRIBUTES>
    ...
    <PROFILE_ELEMENT>
      <ELEMENT_NAME> TARGET_NAME </ELEMENT_NAME>
      <ELEMENT_MEANING> The target_name element provides the names ...
      <VALUE_SYNTAX> ENUMERATION </VALUE_SYNTAX>
      <VALUE_UNIT> N/A </VALUE_UNIT>
      <VALUE_INSTANCE> IDA </VALUE_INSTANCE>
      <VALUE_INSTANCE> JUPITER </VALUE_INSTANCE>
    </PROFILE_ELEMENT>

    <PROFILE_ELEMENT>
      <ELEMENT_NAME> DATA_SET_NAME </ELEMENT_NAME>
      ...
```

Figure 7: PDS DIS Profile

catalogs and gleaned from discipline data models.

The successful location of and supporting interoperability between resources from different domains is strongly dependent on metadata compatibility, or how well the metadata spans the domains. For example, two related domains such as planetary science and astrophysics both associate one or more target bodies with most data products. However, unless the same identifiers are used for a specific target or a mapping between identifiers is determined, the attribute will not support resource location much less interoperability across the domains. In fact as more sophisticated interoperability such as data transformation and correlation is requested, deeper levels of metadata compatibility will be required. For example, once a target body is identified, sufficient metadata must be available for coordinate system conversion.

The resource location service profile is currently being augmented to handle relationships. For

example, within the planetary science model, a data set entity is related to a spacecraft instrument entity through a many-to-many relationship. To describe a relationship, the following meta-attributes are being considered for addition to the profile element section. RELATION_NAME will provide a domain unique identifier for the relationship, RELATION_ELEMENT_NAME will identify the related data element, and RELATION_TYPE will provide the relationship classification. This augmentation will allow additional query constraints for resource profile selection.

The profile, as a set of resource attributes, lends itself in an interesting way to the task of distributed resource location across heterogeneous domains. When considered from an object-oriented perspective, a resource has three modes. These are (1) the description of the resource as represented by the resource's attribute values, (2) the instance of the resource which is obtained by dereferencing the value of the resource's location attribute, and (3) the class definition as represented by the list of resource attributes and data elements in the profile element section.

Within the resource location service, a query can be made for any of the three modes. For example, the primary role of a profile is to provide the location of a candidate resource that can resolve a user query. Once identified, a resource attribute, RESOURCE_LOCATION_ID, is returned. Of course, any other resource attributes such as RESOURCE_TITLE can also be return. As mentioned, the instance of a resource is obtained by de-referencing the value of the RESOURCE_LOCATION_ID attribute. If a resource profile describes an HTML interface, the query could return the actual html page by performing a redirection on the value of RESOURCE_LOCATION_ID. Finally, the class definition of a resource can be returned and used to determine how the class of resources could be queried, even to the extent of dynamically creating a query interface for display to a user.

When using metadata to enable interoperability between domains, the hard problem of finding metadata commonalities across domains arises. This typically involves identifying similar attributes, determining core concepts, possibly generalizing the concept, and determining the key name and aliases. The resource location service has started to address this problem through the use of the data dictionary specialization and the use of the meta-attribute ELEMENT_ALIAS..

The OODT task is keeping abreast of research in the area of metadata development and management including the development of thesauri, ontologies, terminology bases, meta-attribute standards, and tools for metadata management. In particular the concept of terminological ontologies [2] is being considered as a more robust solution for managing metadata commonalities than the simple determination of data element aliases. Simply, this approach focuses on determining the underlying concept with data element names and other attributes managed as concept attributes.

V. Product Service

The product service component, like the profile component, is instantiated as a node in the distributed architecture and provides the capability to return data system products based on a query. This allows each data system to maintain heterogeneous implementations, but still integrate into the enterprise architecture.

Each product server node provides the data access to one or more data systems. A product server node instantiates a Java-based server that integrates with the query service and receives XML-based queries using the XML query structure explained in Section III as part of the Query Service. The product server framework that is provided is a generic Java-based server that dynamically loads query handlers defined and registered with the service. Once a query is received by the framework it then notifies each registered query handler as a separate thread managed by the product server. This allows the product server to time out queries to resources which may not be available. The product server then packages the results from each query handler and returns the results using the XML-defined query definition. These results are then passed back to the Query Service which integrates all the results from the distributed product servers.

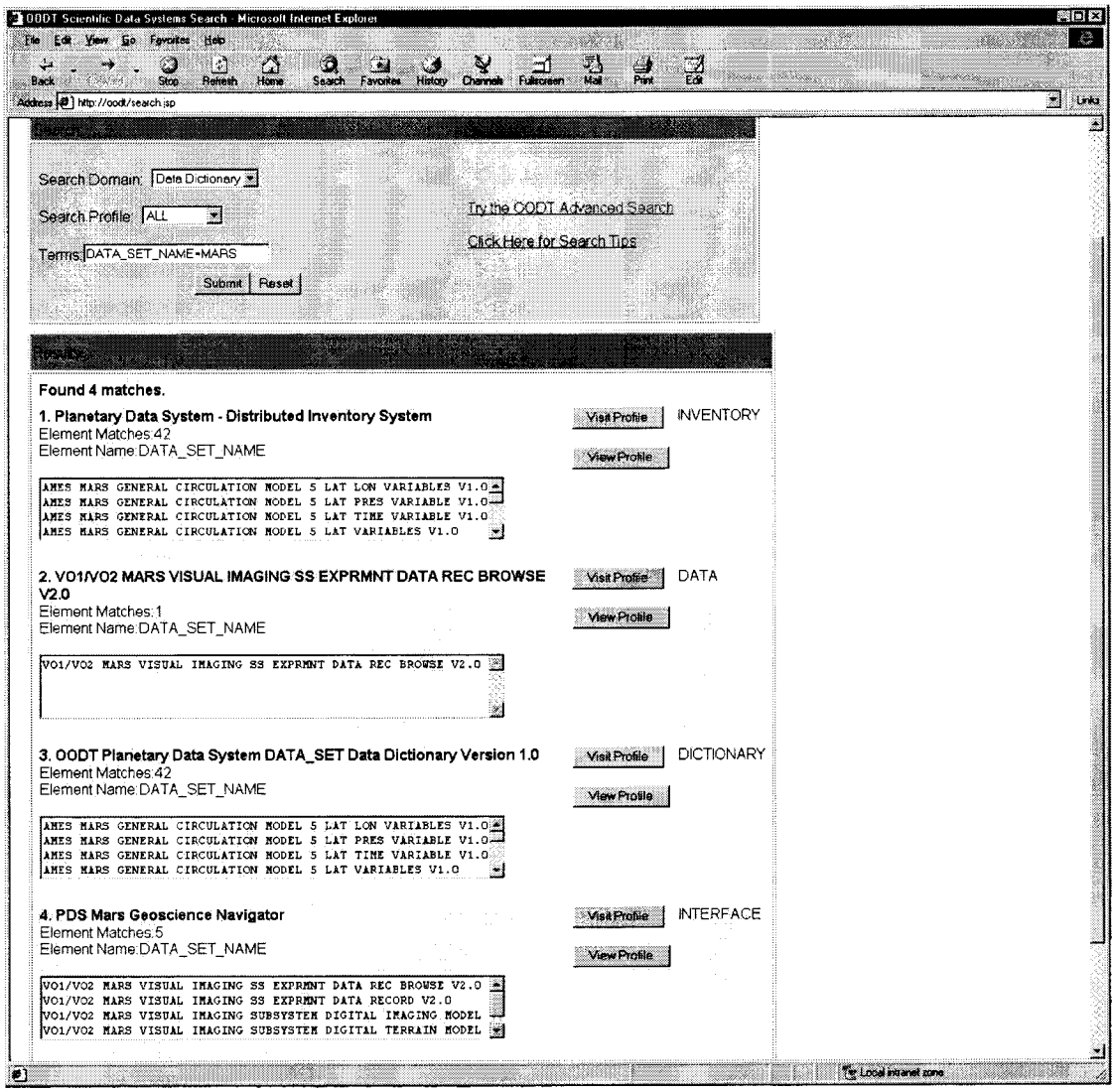

Figure 8. Web interface showing a simple search using the resource location service.

Figure 8 demonstrates a query transaction which returns a list of products that are available from various product servers.

Query handlers provide a wrapper around each data system interface. This abstracts the data system away from the enterprise and allows the query handlers to function as a translation service. Developers implement query handlers using Java's type model, which separates types from classes using interfaces[3]. The query component specifies a standard Java interface to which query handles must conform. Developers creating query handlers define classes that implement the query handler interface allowing the product framework to communicate with the query handlers.

The query handlers are loaded by the product server and passed the XML query. The query handlers transform the queries into the system-dependent query language in order to access the proprietary interface. This moves the responsibility for integrating the data-system–dependent data model onto the data system and away from the OODT data infrastructure. This is an important design consideration for accommodating scalability in a larger enterprise.

[3] An interface in Java is a specification for the methods of a class. A class that implements a named interface must provide a definition for each method specified by interface or else be marked as an abstract class.

An example would be the JPL central PDS node. Implementation of a query handler for this node requires that a mapping between the resource location service XML-based query and the central node's Sybase RDBMS be implemented. The query handler would then translate XML-based queries into a SQL-based query referencing the schema that was implemented by the PDS central node. This then provides the core mapping necessary to allow unique data system products to be retrieved from their native environments.

Once products are received by the query handler they must be transformed into a standard format that can be exchanged. The XML query structure defines the result format which allows for data to be returned in various formats. One of the requirements of this architecture is to provide a list of common interchange formats that imposes a set of standards for interoperability. The challenge is to provide a simple set of common formats for images and text, and require that results that fit into these categories use these formats for interchange. This would mean that all images that are in GIF may need to be converted into JPEG if that was the chosen format for images. It is important to point out that results which do not fit into a these standards can be returned in their native format. The goal is to provide flexibility in the architecture, but where possible promote standards for interoperability.

Product results are returned as ASCII text, or base-64 encoding depending on the data type. Base-64 encoding is used to return complex data products such as images. In many cases data systems may be able to return URL identifiers to data as results, rather than returning the complex data product. This improves performance by limiting the amount of data transported back to the client.

The product server design promotes interoperability by providing an interchange capability to allow a common query mechanism to retrieve products from unique data system implementations. The design presented allows distributed data system nodes to maintain their independence by providing a standard product server that can be extended to access the distributed data systems. This design provides a scalable solution by identifying a standard language for interoperability, and a framework for extending that interoperability to each data system. It also scales by pushing the implementation requirements onto each individual data system.

VI. Conclusion and Future Work

XML has gained in popularity for improving the ability for applications to be integrated through electronic data interchange. The resource location service presented in this paper uses XML for data exchange and metadata definitions as part of its framework for integrating distributed databases across multiple disciplines. This solution allows for loosely related data systems to remain distributed, while providing a content management and interchange capability for locating specific data products and resources archived at remote locations.

The resource location service where possible, promotes the use of open standards. This architecture will accommodate changes as XML and standards for interoperability evolve. Currently, many organizations are looking at standards for electronic data interchange and queries using XML. At time of writing this paper W3C has just published a draft query language for XML.

Metadata is really provides the foundation for our solution. The solution presented, although applied to planetary, astrophysics, and space science data problems, is not limited to those disciplines. In fact, the framework is adaptable based on the metadata definitions that are defined. This allows for the solution to then be applied to other disciplines including healthcare, defense, business, etc. Currently, we are also investigating use of this framework for locating and correlating physiologic and treatment data from pediatric research hospitals distributed across the United States. The benefit of the architecture is that it can easily accommodate different disciplines by refocusing the problem on metadata development. This means that industry and disciplines still must decide on common interchange language that includes common terms, data types, and formats. Once this common language is defined, the resource location service provides the infrastructure necessary for allowing heterogeneous data systems to communicate so that advanced data discovery and mining techniques can be applied and new relationships discovered.

VII. References

Aho, A. V., Hopcroft, J. E., Ullman, J. D. Data Structures and Algorithms. Addison-Wesley 1983.

Booch et al. The Unified Modeling Language User Guide. Addison-Wesley. 1999.

Crichton, D.J., Hughes J.S., Hyon J.J., Kelly, S.C. Object Oriented Data Technology 1999 Annual Report. Interactive Analysis Environments Program. September 1999. http://oodt.jpl.nasa.gov/doc/reports/annual/1999

Data Entity Dictionary Specification Language (DEDSL) - Abstract Syntax, CCSDS 647.0-R-2.0, Draft Recommendation for Space Data System Standards, Consultative Committee on Space Data Systems, November 1999.

Devlin, B. Data Warehouse from Architecture to Implementation. Addison-Wesley. 1997.

Elmasri,R., Navathe,S., Fundamentals of Database Systems. The Benjamin/Cumings Publishing Company, Inc. 1994.

Hughes,J.S., Crichton,D.J., Hyon,J.J., Kelly,S.C., A Multi-Discipline Metadata Registry for Science Interoperability, Open Forum on Metadata Registries, ISO/IEC JTC1/SC32, Data Management and Interchange, January 2000, http://www.sdct.itl.nist.gov/~ftp/l8/sc32wg2/2000/events/openforum/index.htm

Hovy, E. Using Ontologies to Enable Access to Multiple Heterogeneous Databases, Open Forum on Metadata Registries, ISO/IEC JTC1/SC32, Data Management and Interchange, January 2000, http://www.sdct.itl.nist.gov/~ftp/l8/sc32wg2/2000/events/openforum/index.htm

Deutsch, A., Fernadez, M., Florescu, D., Levy, A., Suciu, D. XML-QL: A Query Language for XML. Submitted to W3C August 19, 1998. http://www.w3.org/TR/NOTE-xml-ql

Maruyama, H., Tamura, K., Uramoto, N. XML and Java: Developing Web Applications. Addison-Wesley. 1999.

Object Management Group. CORBA/IIOP 2.3.1 Specification. October 1999.

Orbacus for C++ and Java version 3.1.3. Object Oriented Concepts, Inc. 1999. http://ooc.com/

W3C. Document Object Model (DOM), Level 2 Specification. http://www.w3.org/TR/1999/CR-DOM-Level-2-19991210.

W3C. Extensible Markup Language (XML), Version 1.0. http://www.w3.org/TR/1998/REC-xml-19980210.

W3C. XML Query Requirements. W3C Working Draft. 31 January 2000. http://www.w3.org/TR/2000/WD-xmlquery-req-20000131

VIII. About the Authors

Daniel Crichton is a Project Element Manager at JPL, and the principal investigator for the Object Oriented Data Technology task where he is leading a research task developing distributed frameworks for integrating science data management and archiving systems. He also currently serves as the implementation manager and architect of a JPL initiative to build an enterprise data architecture. His interests are in distributed architectures, enterprise and Internet technologies, and database systems. He holds a B.S. and M.S. in Computer Science. He can be reached at Daniel.J.Crichton@jpl.nasa.gov

Steven Hughes is a System Engineer at JPL, and a Co-Investigator for the Object Oriented Data Technology task. He is currently the technical lead engineer for the Planetary Data System and was instrumental in the development of the planetary science meta-model. His interests are in distributed architectures and the role of metadata in interoperability. He holds a B.S. and M.S. in Computer Science. He can be reached at Steven.Hughes@jpl.nasa.gov

Jason Hyon has been with JPL since 1985 and is currently a group supervisor for the Information Management Technology group. His research focuses on real time data architecture, information management, object-oriented distributed computing, and optical data storage. He is a principal investigator and a manager for

the Integrated Information Management task for the JPL TMOD on-board data management research. He manages the Data Archival and Retrieval Enhancement (DARE) task for Defense Threat Reduction Agency, and NASA's Regional Planetary Imaging Facility. He is also a co-investigator for NASA's Data Distribution Lab and for the Object Oriented Data Technology task. He can be reached at Jason.Hyon@jpl.nasa.gov

Sean Kelly is a Senior Software Engineer at User Technology Associates and a consultant to JPL. He is currently supporting implementation for the Object Oriented Data Technology task. His interests include practical applications of software methods and leveraging web technologies in unique ways. He holds a B.S. in Computer Science and a B.S. in Technical Communication. He can be reached at Sean.Kelly@jpl.nasa.gov